Discrete Applied Mathematics 26 (1990) 235–253
North-Holland


# A SURVEY OF ALGORITHMS FOR THE SINGLE MACHINE TOTAL WEIGHTED TARDINESS SCHEDULING PROBLEM

T.S. ABDUL-RAZAQ
*Department of Mathematics, Basra University, Iraq*

C.N. POTTS
*Faculty of Mathematical Studies, University of Southampton, Southampton SO9 5NH, UK*

L.N. Van WASSENHOVE
*Econometric Institute, Erasmus University Rotterdam, Rotterdam, Netherlands*

Received 25 April 1988
Revised 31 January 1989

This paper surveys algorithms for the problem of scheduling jobs on a single machine to minimize total weighted tardiness. Special attention is given to two dynamic programming and four branch and bound algorithms. The dynamic programming algorithms both use the same recursion defined on sets of jobs, but they generate the sets in lexicographic order and cardinality order respectively. Two of the branch and bound algorithms use the quickly computed but possibly rather weak lower bounds obtained from linear and exponential functions of completion times problems. These algorithms rely heavily on dominance rules to restrict the search. The other two branch and bound algorithms use lower bounds obtained from the Lagrangean relaxation of machine capacity constraints and from dynamic programming state-space relaxation. They invest a substantial amount of computation time at each node of the search tree in an attempt to generate tight lower bounds and thereby generate only small search trees. A computational comparison of all these algorithms on problems with up to 50 jobs is given.

## 1. Introduction

The single machine total weighted tardiness problem may be stated as follows. Each of $n$ jobs (numbered $1, \ldots, n$) is to be processed without interruption on a single machine that can handle only one job at a time. Job $i$ ($i = 1, \ldots, n$) becomes available for processing at time zero, requires an integer processing time $p_i$, and has a positive weight $w_i$ and a due date $d_i$. For a given processing order of the jobs the (earliest) completion time $C_i$ and the tardiness $T_i = \max\{C_i - d_i, 0\}$ of job $i$ ($i = 1, \ldots, n$) can be computed. The objective is to find a processing order of the jobs that minimizes the total weighted tardiness $\sum_{i=1}^{n} w_i T_i$.

Emmons [5] derives several dominance rules that restrict the search for an optimal solution to the total weighted tardiness problem. These rules are used in both dynamic programming and branch and bound algorithms. Branch and bound approaches can be loosely classified into those that invest a substantial amount of

computation time to obtain tight lower bounds and those which use quickly computed, but possibly rather weak, lower bounds. In the latter case, algorithms rely heavily on Emmons' rules to restrict the search.

In this paper we give a survey of dynamic programming and branch and bound algorithms. Where no computational experience for a particular algorithm is reported in the literature, we perform comparative computational tests to assess its effectiveness. Our survey ignores the very early branch and bound algorithms in which the lower bounds are very weak. For instance, the algorithms proposed by Elmaghraby [4] in which the lower bound is obtained from the tardiness of the job scheduled last and by Shwimer [21] in which the lower bound is obtained from the tardiness of the job scheduled last and the maximum tardiness of the remaining jobs are shown by Rinnooy Kan et al. [17] to require very large computation times.

This paper is organized as follows. In Section 2, Emmons' dominance rules are described. Section 3 gives a general description of the dynamic programming approach for solving the problem and provides details of two algorithms. The various lower bounding schemes that can be used in branch and bound algorithms are described next in Section 4. Section 5 describes the branching rule, search strategy and dominance rules that are used in the branch and bound algorithms. Section 6 reports on computational experience with the dynamic programming and branch and bound algorithms. Some concluding remarks are given in Section 7.

## 2. Dominance rules

Emmons' dominance rules are described in this section. They play a major role in the algorithms that are described later. Suppose that the rules have already been applied to yield, for each job $h$, a set $B_h$ and a set $A_h$ of jobs which precede and succeed job $h$ in at least one optimal sequence. Let $S$ denote the set of all jobs.

**Dominance Theorem.** *There exists an optimal sequence in which job i is sequenced before job j if one of the following conditions is satisfied.*

(a) $p_i \leq p_j$, $w_i \geq w_j$ *and* $d_i \leq \max\{d_j, \sum_{h \in B_j} p_h + p_j\}$;

(b) $w_i \geq w_j$, $d_i \leq d_j$ *and* $d_j \geq \sum_{h \in S - A_i} p_h - p_j$;

(c) $d_j \geq \sum_{h \in S - A_i} p_h$.

Elmaghraby's lemma [4] follows from condition (c): If a job $j$ with $d_j \geq \sum_{h \in S} p_h$ is found, then there exists an optimal sequence in which this job is sequenced last. In such a case job $j$ is removed from the problem.

Whenever jobs $i$ and $j$ are found satisfying the conditions of the Dominance Theorem, an arc $(i,j)$ is added to *precedence graph* $G_P$ together with any other arcs $(h,k)$ that are implied by transitivity. We refer to $h$ as a *predecessor* of $k$ and to $k$ as a *successor* of $h$. The procedure is repeated until no further arcs can be added to $G_P$. At this stage, any job which is a successor of all remaining jobs is removed

from the problem. Similarly, any job $i$ which is a predecessor of all remaining jobs is removed from the problem and the due dates of remaining jobs are reduced by $p_i$.

Consider the transitive reduction of $G_P$ which is obtained by removing all arcs of $G_P$ that are implied by transitivity. For each arc $(i,j)$ of the transitive reduction of $G_P$, $i$ is an *immediate predecessor* of $j$ and $j$ is an *immediate successor* of $i$. Using $G_P$, the *earliest completion time* $C_i^E = \sum_{h \in B_i} p_h + p_i$ and the *latest completion time* $C_i^L = \sum_{h \in S - A_i} p_h$ of job $i$ $(i = 1, \ldots, n)$ are computed.

In many applications, including those that follow in the next section, it is convenient to regard $G_P$ as representing *precedence constraints* on the jobs that must be satisfied.

Emmons proposes a search algorithm based on his dominance rules. Whenever the rules fail to generate a complete sequence, jobs $i$ and $j$ are suitably chosen and the possibilities that either $i$ is a predecessor or a successor of $j$ are examined. For each possibility the dominance rules are reapplied and the process continues until a complete sequence is generated. Computational tests by Baker and Martin [2] with this approach indicate that it is inferior to dynamic programming and branch and bound algorithms.

## 3. Dynamic programming algorithms

### *3.1. Recursion equations*

All the well-known dynamic programming algorithms use the same dynamic programming recursion equations which are as follows. Let $f(R,j)$ be the minimum total weighted tardiness when the jobs of the set $R - \{j\}$ are sequenced in the first $r-1$ positions followed by job $j$ in position $r$, where $r = |R|$. Then we may define $f(R,*) = \min_{j \in R}\{f(R,j)\}$ as the minimum total weighted tardiness when the jobs of $R$ are sequenced in the first $r$ positions. For this formulation, the objective is to find $f(S,*)$ (recall $S = \{1, \ldots, n\}$) using the recursion equations

$$f(R,j) = \min_{i \in R - \{j\}} \left\{ f(R - \{j\}, i) + w_j \max\left\{ \sum_{k \in R} p_k - d_j, 0 \right\} \right\} \tag{1}$$

that are initialized by setting $f(\emptyset, j) = 0$ $(j = 1, \ldots, n)$. Clearly, recursion (1) may be written in the equivalent, more usual form, in which the objective is to find $f(S,*)$ from the recursion equations

$$f(R,*) = \min_{j \in R} \left\{ f(R - \{j\}, *) + w_j \max\left\{ \sum_{k \in R} p_k - d_j, 0 \right\} \right\} \tag{2}$$

that are initialized by setting $f(\emptyset, *) = 0$. Equations (2) define a *forward recursion* as an initial partial sequence corresponds to each pair $R$ and $f(R,*)$. A *backward recursion* is derived as follows. Let $f'(\bar{R},*)$ be the minimum total weighted tardiness when

the jobs of the set $\bar{R}$ are sequenced in the final $\bar{r}$ positions, where $\bar{r} = |\bar{R}|$. We can regard $\bar{R}$ as the complement of the set $R$ which occurs in (2); i.e., $\bar{R} = S - R$. Then the problem is to find $f'(S, *)$ from the recursion equations

$$f'(\bar{R}, *) = \min_{j' \in \bar{R}} \left\{ f'(\bar{R} - \{j'\}, *) + w_{j'} \max \left\{ \sum_{k \in S - \bar{R}} p_k + p_{j'} - d_{j'}, 0 \right\} \right\} \tag{3}$$

that are initialized by setting $f'(\emptyset, *) = 0$.

As mentioned in the previous section, the graph $G_P$, obtained by applying Emmons' dominance rules, is assumed to define precedence constraints on the jobs. As a consequence, equations (2) and (3) are modified as follows. Firstly, only *feasible sets* $R$ and $\bar{R}$ are considered in (2) and (3) respectively, i.e., sets $R$ for which all predecessors of jobs of $R$ are contained in $R$ and sets $\bar{R}$ for which all successors of jobs of $\bar{R}$ are contained in $\bar{R}$. Clearly, there is a one to one correspondence between the feasible sets $R$ for (2) and the feasible sets $\bar{R} = S - R$ for (3). Secondly, the minimization in (2) is restricted to jobs $j$ such that $R - \{j\}$ is a feasible set, i.e., only jobs $j$ which have no successors in $R$ are considered. Similarly, the minimization in (3) is restricted to jobs $j'$ which have no predecessors in $\bar{R}$.

In addition to Emmons' rules, Elmaghraby's lemma may be applied at each stage of the dynamic programming recursion to improve efficiency. Consider a set $R$ for which $j \in R$ with $d_j \geq \sum_{h \in R} p_h$ exists. If the jobs of $R$ are sequenced in the initial positions, then, from the lemma, there exists an optimal ordering of these jobs in which job $j$ is sequenced last. In such a case, recursion (2) reduces to $f(R, *) = f(R - \{j\}, *)$ which involves no minimization and thus reduces computation. Suppose that $\bar{R} = S - R$, where again $j \in R$ with $d_j \geq \sum_{h \in R} p_h$ exists. If there exists an optimal schedule in which the jobs of $\bar{R}$ are sequenced in the final positions, then, from the lemma, there exists an optimal schedule in which the jobs of $\bar{R} \cup \{j\}$ are sequenced in the final positions. Thus, sets $\bar{R} \cup \{k\}$ $(k \in S - \bar{R};\ k \neq j)$ may be regarded as infeasible thereby reducing the total number of feasible sets and, consequently, storage requirements.

The dynamic programming algorithms differ mainly in the order in which the feasible sets $R$ (or $\bar{R}$) are generated and the way in which the values $f(R, *)$ (or $f'(\bar{R}, *)$) are stored. The algorithms of Schrage and Baker [20] and Lawler [13] are described below. The approach of Srinivasan [23] is similar to that of Lawler, although the implementation described by Lawler makes his algorithm far more effective. Thus, Srinivasan's algorithm is not considered separately.

### 3.2. *The Schrage–Baker algorithm*

In the Schrage–Baker dynamic programming algorithm the feasible sets $R$ are generated in *lexicographic order*, i.e., in increasing order of $\sum_{i \in R} 2^{i-1}$. Also, an integer label is assigned to each job (the label given to a job is chosen so that it exceeds by one the sum of labels already assigned to jobs that are neither its predecessors nor its successors) so that each feasible set is given an address which is equal to the

sum of labels of jobs in that set. The value $f(R, *)$ is stored in a location corresponding to the address for the set $R$. It is often the case that there are addresses for which there is no feasible set in which case storage space is wasted. The storage space required by the algorithm is equal to the sum of all labels and is known before any of the recursion equations are solved. Also, since all the values $f(R, *)$ are stored, once all recursion equations have been solved, a simple backtracking procedure allows the optimal sequence to be found. (Kao and Queyranne [11] give an alternative implementation in which the storage space is used cyclically to give a storage requirement which is equal to the maximum label: backup storage is used to find the optimal sequence.) If the sum of labels does not exceed the number of words of computer storage available (48 000 in our computational experiments), then the problem is solved; otherwise computation is abandoned for that problem.

### *3.3. Lawler's algorithm*

The algorithm described here is a variant of the dynamic programming algorithm of Lawler [13] which is designed so that core storage requirements are kept to a minimum. Further to this aim, a backward recursion is used which allows Elmaghraby's lemma to be applied dynamically thereby reducing the number of feasible sets that need to be considered. The feasible sets $\bar{R}$ are generated in *cardinality order*. More precisely, when all feasible sets $\bar{R}$ of cardinality $\bar{r}$ have been generated, all feasible sets of cardinality $\bar{r}+1$ are of the form $\bar{R} \cup \{k\}$ where $k \notin \bar{R}$ and where all successors of $k$ are in $\bar{R}$. Each set is represented by its incidence vector, where the incidence vector for set $\bar{R}$ is defined as $\sum_{i \in \bar{R}} 2^{n-i}$. If the incidence vectors for the sets of successors of each job $k$ are stored, the tests of whether $k \notin \bar{R}$ and whether all successors of $k$ are in $\bar{R}$ can be performed in constant time provided that $n$ does not exceed the computer word length. (In FORTRAN, this is achieved by performing a logical AND statement on the logical variables obtained from the incidence vectors through an EQUIVALENCE statement.) During the generation of feasible sets of cardinality $\bar{r}+1$ from a list of feasible sets $\bar{R}$ of cardinality $\bar{r}$ that are stored in increasing order of incidence vectors, if $d_j \geq \sum_{i \in S-\bar{R}} p_i$ for some $j \in S-\bar{R}$, then Elmaghraby's lemma eliminates sets $\bar{R} \cup \{k\}$, where $k \neq j$. The generation process starts by forming the list of feasible sets of the form $\bar{R} \cup \{1\}$ together with the corresponding total weighted tardiness values $f'(\bar{R}, *) + w_1 \max\{\sum_{i \in S-\bar{R}} p_i - d_1, 0\}$. The next step is to find feasible sets of the form $\bar{R} \cup \{2\}$ and compute their total weighted tardiness values. This list is merged with the list of feasible sets $\bar{R} \cup \{1\}$: when duplicate sets (i.e., sets with the same incidence vector) are found during the merge, only the entry with the smaller total weighted tardiness value is retained. Note that the list of feasible sets of the form $\bar{R} \cup \{2\}$ does not need to be constructed explicitly since forward pointers to the appropriate entries in the list of feasible sets of cardinality $\bar{r}$ allow the necessary information to be accessed. Feasible sets of the form $\bar{R} \cup \{3\}, \ldots, \bar{R} \cup \{n\}$ are successively created and merged to give a complete list of feasible sets of cardinality $\bar{r}+1$ stored in increasing order of incidence vectors.

At this stage, the list of sets of cardinality $\bar{r}$ is discarded and the process of generating sets of cardinality $\bar{r}+2$ commences.

A natural implementation is to use one word of storage for each incidence vector. The total weighted tardiness value occupies a second word and the processing time $\sum_{i \in S-\bar{R}} p_i$ occupies a third word. Thus, the maximum number of sets that can be stored simultaneously cannot exceed one third of the number of words of computer storage available (16 000 in our computational experiments). The storage space required by the algorithm needs to be sufficient to store all sets of cardinality $\bar{r}$ and all sets of cardinality $\bar{r}+1$ simultaneously ($\bar{r}=1,\ldots,n-1$); if it is insufficient, then computation is abandoned for that problem. It is a disadvantage of the algorithm that the storage requirements of a particular problem cannot be predicted before any recursion equations are solved. Another disadvantage is that it is not simple to find the optimal sequence after the recursion equations have been solved unless, as is often the case, backup storage is used.

## 4. Lower bounds

### *4.1. Formulation as a transportation problem*

Lawler [12] shows that a relaxation of the problem can be formulated as a transportation problem if job $i$ ($i=1,\ldots,n$) is split into $p_i$ pieces each having a processing time of one unit. There are now $P=\sum_{i=1}^{n} p_i$ unit time pieces. We define variables $x_{it}$ ($i=1,\ldots,n$; $t=1,\ldots,P$) by

$$x_{it}=\begin{cases}1, & \text{if a piece of job } i \text{ is scheduled in the interval } [t-1,t],\\ 0, & \text{otherwise.}\end{cases}$$

By relaxing the constraints that the unit time pieces of each job are to be sequenced in adjacent time intervals, a lower bound is obtained from the transportation problem

$$\text{minimize} \quad \sum_{i=1}^{n}\sum_{t=1}^{P} b_{it}x_{it},$$

$$\text{subject to} \quad \sum_{t=1}^{P} x_{it}=p_i, \quad i=1,\ldots,n,$$

$$\sum_{i=1}^{n} x_{it}=1, \quad t=1,\ldots,P,$$

$$x_{it}\in\{0,1\}, \quad i=1,\ldots,n;\ t=1,\ldots,P,$$

where $b_{it}$ is an appropriately chosen cost. Using Emmons' rules, we may assume that all processing on job $i$ ($i=1,\ldots,n$) is performed in intervals $[t-1,t]$ for $t=C_i^E-p_i+1,\ldots,C_i^L$. To force processing to be executed only in these time intervals,

we set $b_{it} = \infty$ for $t = 1, \ldots, C_i^E - p_i, C_i^L + 1, \ldots, P$. If this transportation problem is to provide an optimal solution of the original $n$-job problem when the unit time pieces of each job are sequenced in adjacent time intervals, then we require for each job $i$ that $\sum_{t'=t-p_i+1}^{t} b_{it'} = w_i \max\{t - d_i, 0\}$ for $t = C_i^E - p_i + 1, \ldots, C_i^L$. However, a lower bound is obtained by using costs $b'_{it}$, where $b'_{it} \leq b_{it}$, which satisfy $\sum_{t'=t-p_i+1}^{t} b'_{it'} \leq w_i \max\{t - d_i, 0\}$ for $t = C_i^E - p_i + 1, \ldots, C_i^L$. One choice of $b'_{it}$ which satisfies these conditions is

$$b'_{it} = \begin{cases} w_i \max\{t - d_i, 0\}/p_i, & \text{for } t = C_i^E - p_i + 1, \ldots, C_i^L, \\ \infty, & \text{otherwise.} \end{cases}$$

The lower bound based on the solution of the transportation problem with costs $b'_{it}$ is computed in $O(n^2 P)$ time.

Gelders and Kleindorfer [8,9] are responsible for the development of this approach. They explore the possibility of obtaining a lower bound through the use of dual variables computed at a previous node of the branch and bound search tree. In a further attempt to reduce computation time for lower bounds, they explore the use of time intervals which are longer than one unit.

### *4.2. Formulation as an assignment problem*

Rinnooy Kan et al. propose the following lower bound that is obtained from the solution of an assignment problem. If it is possible to compute a cost $c_{ij}$ $(i, j = 1, \ldots, n)$ representing the weighted tardiness of job $i$ if it is sequenced in position $j$, then an optimal sequence is obtained from the solution of the linear assignment problem based on these costs. In general, however, the weighted tardiness depends on the jobs which are scheduled previously, so $c_{ij}$ cannot be computed. Nevertheless, by solving an assignment problem based on costs $c'_{ij}$, where $c'_{ij} \leq c_{ij}$, a lower bound is obtained. To obtain a lower bound on the weighted tardiness of job $i$ when it is sequenced in position $j$, we assume Emmons' rules define precedence constraints on the jobs. Thus, for $j = |B_i| + 1, \ldots, |S - A_i|$, the earliest completion time of job $i$ if it is sequenced in position $j$ is obtained by summing the processing times of the jobs of $B_i$, of job $i$ and of another $j - |B_i| - 1$ jobs. This earliest completion time is

$$C_{ij}^E = C_i^E + \min\left\{ \sum_{h \in Q} p_h \;\middle|\; Q \subseteq S - A_i - B_i - \{i\};\ |Q| = j - |B_i| - 1 \right\}.$$

Therefore, we obtain

$$c'_{ij} = \begin{cases} w_i \max\{C_{ij}^E - d_i, 0\}, & \text{for } j = |B_i| + 1, \ldots, |S - A_i|, \\ \infty, & \text{otherwise.} \end{cases} \tag{4}$$

The lower bound based on the solution of the assignment problem with costs $c'_{ij}$ is computed in $O(n^3)$ time.

Picard and Queyranne [14] generalize the assignment approach by using a time dependent travelling salesman problem formulation. Let job 0 be a dummy job with $p_0 = w_0 = d_0 = 0$ which is to be sequenced first. These generalized bounds consider a cost $c_{hij}$ $(h = 0, \ldots, n;\ i, j = 1, \ldots, n)$ which represents the weighted tardiness of job $i$ when it is sequenced in position $j+1$ immediately after job $h$. The objective is to find a sequence $(\sigma(1), \ldots, \sigma(n))$ of the jobs $1, \ldots, n$ which minimizes the total cost $\sum_{j=1}^{n} c_{\sigma(j-1),\sigma(j),j}$, where $\sigma(0) = 0$. As in the assignment formulation, costs $c_{hij}$ cannot be computed, so lower bound values $c'_{hij} \le c_{hij}$, obtained from an equation similar to (4) (but using the information that job $h$ is forced to be a predecessor of job $i$) are used. Although the time dependent travelling salesman problem cannot be solved efficiently, Picard and Queyranne propose a lower bound based on a shortest path relaxation. This lower bound can be derived using the dynamic programming state-space relaxation method [3].

### *4.3. Reduction of the total weighted tardiness*

Although the Potts–Van Wassenhove lower bound [15] is originally obtained using Lagrangean relaxation, it may also be derived by reducing the objective to a linear function as follows. Clearly, for job $i$ $(i = 1, \ldots, n)$ we have

$$w_i T_i = w_i \max\{C_i - d_i, 0\} \ge u_i \max\{C_i - d_i, 0\} \ge u_i (C_i - d_i),$$

where $w_i \ge u_i \ge 0$. Let $u = (u_1, \ldots, u_n)$ be a vector of linear weights (i.e., weights for the linear functions $C_i - d_i$) chosen so that $0 \le u_i \le w_i$ $(i = 1, \ldots, n)$. Then a lower bound is given by the linear function

$$\mathrm{LB}_{\mathrm{LIN}}(u) = \min\left\{\sum_{i=1}^{n} u_i (C_i - d_i)\right\} \le \min\left\{\sum_{i=1}^{n} w_i T_i\right\}.$$

This shows that the solution of a total weighted completion time problem provides a lower bound on the total weighted tardiness problem. Given $u$, the weighted completion time problem is solved by Smith's shortest weighted processing time rule [22] in which jobs are sequenced in nonincreasing order of $u_i/p_i$.

An alternative lower bound is now derived by reducing the weighted tardiness to an exponential function rather than the linear function used above. In this derivation, it is assumed that $P = \sum_{j=1}^{n} p_j > d_i$ for each job $i$. (If $P \le d_i$, then job $i$ is sequenced last by Elmaghraby's lemma and discarded.) For any positive $\alpha$ we aim to find a nonnegative weight $v_i$ for job $i$ $(i = 1, \ldots, n)$ that satisfies

$$w_i T_i = w_i \max\{C_i - d_i, 0\} \ge v_i (\mathrm{e}^{\alpha(C_i - d_i)} - 1). \tag{5}$$

Clearly, (5) holds when $C_i \le d_i$. When $d_i < C_i \le P$, then (5) also holds provided that

$$v_i \le w_i (P - d_i)/(\mathrm{e}^{\alpha(P - d_i)} - 1). \tag{6}$$

Let $v = (v_1, \ldots, v_n)$ be a vector of nonnegative exponential weights (i.e., weights for the exponential functions $\mathrm{e}^{\alpha(C_i - d_i)} - 1$) which satisfies (6). Then, if $\alpha$ is chosen, a

lower bound is given by the exponential function

$$\mathrm{LB}_{\mathrm{EXP}}(\upsilon)=\min\left\{\sum_{i=1}^{n}\upsilon_i(\mathrm{e}^{\alpha(C_i-d_i)}-1)\right\}\leq\min\left\{\sum_{i=1}^{n}w_iT_i\right\}.$$

This shows that the solution of a total weighted exponential function of completion time problem provides a lower bound on the total weighted tardiness problem. When $\alpha$ and $\upsilon$ are known, the total weighted exponential completion time problem is solved by Rothkopf's rule [18] in which jobs are sequenced in nonincreasing order of $\upsilon_i/(\mathrm{e}^{\alpha d_i}(1-\mathrm{e}^{-\alpha p_i}))$.

Ideally, nonnegative values of $u$ and $\upsilon$ would be selected to maximize $\mathrm{LB}_{\mathrm{LIN}}(u)$ and $\mathrm{LB}_{\mathrm{EXP}}(\upsilon)$ subject to $u_i\leq w_i$ and (6) for each job $i$. To obtain these best possible bounds, the subgradient optimization method [7, 10] could be used to find $u$ and $\upsilon$. However, since it is computationally expensive to apply, the following noniterative heuristic method of Potts and Van Wassenhove [15] to determine $u$ and $\upsilon$ provides an attractive alternative.

Suppose that a heuristic method is first applied to obtain a sequence and job completion times $C_i^H$ $(i=1,\ldots,n)$. Suppose also that the jobs are renumbered so that the heuristic sequence is $(1,\ldots,n)$. Then the vector of linear weights $u$ is chosen to maximize $\mathrm{LB}_{\mathrm{LIN}}(u)$, subject to the condition that the heuristic sequence is an optimal solution of the total weighted completion time problem. Similarly, $\upsilon$ is chosen to maximize $\mathrm{LB}_{\mathrm{EXP}}(\upsilon)$ subject to the condition that the heuristic sequence is an optimal solution of the total weighted exponential function of completion times problem. A linear programming problem $(P)$ of the form

$$(P)\qquad \begin{aligned} &\text{maximize} && \mathrm{LB}(z)=\sum_{i=1}^{n}a_iz_i,\\ &\text{subject to} && b_iz_i\geq b_{i+1}z_{i+1}, \quad i=1,\ldots,n-1, && (7)\\ &&& 0\leq z_i\leq c_i, \qquad i=1,\ldots,n, && (8)\end{aligned}$$

where $a_i$ is a constant and $b_i$ and $c_i$ are nonnegative constants $(i=1,\ldots,n)$, can be solved to find $u$ and $\upsilon$. When $a_i=C_i^H-d_i$, $b_i=1/p_i$, $c_i=w_i$ and $z_i=u_i$, the solution of problem $(P)$ yields the lower bound $\mathrm{LB}_{\mathrm{LIN}}$. Similarly, when $a_i=\mathrm{e}^{\alpha(C_i^H-d_i)}-1$, $b_i=1/(\mathrm{e}^{\alpha d_i}(1-\mathrm{e}^{-\alpha p_i}))$, $c_i=w_i(P-d_i)/(\mathrm{e}^{\alpha(P-d_i)}-1)$ and $z_i=\upsilon_i$, the solution of problem $(P)$ yields the lower bound $\mathrm{LB}_{\mathrm{EXP}}$.

We describe next an algorithm which solves problem $(P)$ in linear time. Firstly, we observe that for any jobs $h$ and $i$ where $h\leq i$, constraints (7) and (8) yield

$$b_iz_i\leq b_hz_h\leq b_hc_h. \tag{9}$$

Let us define

$$c_i'=\min_{h\in\{1,\ldots,i\}}\{b_hc_h/b_i\}, \quad i=1,\ldots,n.$$

In view of (9), adding the constraints

$$0 \leq z_i \leq c_i', \quad i = 1, \ldots, n, \tag{10}$$

to problem $(P)$ does not alter its solution. Since $c_i' \leq c_i$ $(i = 1, \ldots, n)$, these new constraints imply the original constraints (8) which may therefore be dropped.

The algorithm to solve problem $(P)$ is a generalization of the one which is used by Potts and Van Wassenhove [15] to obtain $LB_{LIN}$ and their proof of optimality can be generalized. In the algorithm, the variable $D$ indicates whether $z_k$ is set to its lower bound value given by (7) or its upper bound value given by (10) and the variable LB provides the lower bound.

**Algorithm LP**

*Step* 1. Set $D = 0$, $LB = 0$ and $k = 1$.
*Step* 2. Set $D = D + a_k/b_k$. If $D \leq 0$, go to Step 4.
*Step* 3. Set $LB = LB + Db_k c_k'$ and set $D = 0$.
*Step* 4. If $k = n$, stop. Otherwise set $k = k + 1$ and go to Step 2.

Clearly, the time requirement for solving problem $(P)$ using Algorithm LP is $O(n)$.

To obtain the lower bounds $LB_{LIN}$ and $LB_{EXP}$, a heuristic method is required to sequence the jobs. An obvious way to implement this lower bounding scheme in a branch and bound algorithm is to use a heuristic method at the root node of the branch and bound search tree to generate an initial sequence. Thereafter, the sequence which currently corresponds to the best solution found by the branch and bound algorithm is used to generate the lower bounds $LB_{LIN}$ or $LB_{EXP}$.

To find the value of $\alpha$ required for $LB_{EXP}$, we suggest that a golden section search is performed to find a "good" value $\alpha^*$ at the root node of the search tree. A heuristic method can then be used to find a value $\alpha$ to be used within the search tree since it is computationally too expensive to apply the golden section search at each node. Based on the results of numerous tests with various rules to determine $\alpha$, the following heuristic gives satisfactory results. For $\alpha^* < 0.0001$ set $\alpha = 1000\alpha^*$ and for $\alpha^* \geq 0.0001$ set $\alpha = 6\alpha^*/(5RDD + TF - 1)$ if $5RDD + TF > 1.5$ and $\alpha = 12\alpha^*$ if $5RDD + TF \leq 1.5$, where $RDD = (\max_{i=1,\ldots,n}\{d_i\} - \min_{i=1,\ldots,n}\{d_i\})/\sum_{i=1}^{n} p_i$ is the relative range of due dates and $TF = 1 - \sum_{i=1}^{n} d_i/(n\sum_{i=1}^{n} p_i)$ is the average tardiness factor.

### *4.4. Lagrangean relaxation*

In this section, Fisher's lower bound [6] is described. It is obtained by performing a Lagrangean relaxation of the machine capacity constraints which restrict the machine to process only one job at a time. In this approach, a multiplier is associated with each of the $P = \sum_{i=1}^{n} p_i$ unit time intervals during which the machine is required to be busy. The problem may be formulated using the zero-one

variables $y_{it}$ $(i=1,\ldots,n;\ t=1,\ldots,P)$ where

$$y_{it}=\begin{cases}1, & \text{if the machine processes job } i \text{ in the time interval } [t-1,t],\\ 0, & \text{otherwise.}\end{cases}$$

If we use Emmons' rules, the problem may be stated as

$$\text{maximize} \quad \sum_{i=1}^{n} w_i \max\{C_i-d_i,0\},$$

$$\text{subject to} \quad C_i \in \{C_i^E,\ldots,C_i^L\}, \qquad i=1,\ldots,n,$$

$$y_{it}=1 \text{ for } t=C_i-p_i+1,\ldots,C_i, \qquad i=1,\ldots,n, \tag{11}$$

$$y_{it}=0 \text{ for } t=1,\ldots,C_i-p_i,C_i+1,\ldots,P, \qquad i=1,\ldots,n, \tag{12}$$

$$\sum_{i=1}^{n} y_{it}=1, \qquad t=1,\ldots,P, \tag{13}$$

$$C_i+p_j \le C_j \text{ for each arc } (i,j) \text{ of } G_P. \tag{14}$$

A lower bound is obtained by performing a Lagrangean relaxation of constraints (13) and relaxing some of constraints (14). Applying (11) and (12), the resulting bound is given by

$$\mathrm{LB}_{\mathrm{LR}}(\mu)=\min\left\{\sum_{i=1}^{n}\left(w_i\max\{C_i-d_i,0\}+\sum_{t=C_i-p_i+1}^{C_i}\mu_t\right)\right\}-\sum_{t=1}^{P}\mu_t,$$

$$\text{subject to} \quad C_i \in \{C_i^E,\ldots,C_i^L\}, \qquad i=1,\ldots,n,$$

$$C_i+p_j \le C_j \text{ for each arc } (i,j) \text{ of } G_P',$$

where $\mu=(\mu_1,\ldots,\mu_P)$ is a vector of multipliers corresponding to constraints (13) and $G_P'$ is a subgraph of $G_P$. To enable the Lagrangean problem to be efficiently solved, $G_P'$ is obtained by deleting arcs from $G_P$ until each job has at most one immediate successor in the resulting graph. Arcs are deleted so that longest paths in the network are retained. It is assumed that $G_P'$ is connected. If not, a dummy job $n+1$ with a large due date is added to generate arcs $(i,n+1)$ for $i=1,\ldots,n$, that ensure that $G_P'$ is connected.

We discuss next the problem of solving this Lagrangean problem for a given $\mu$ to obtain the lower bound. Let $\beta_i$ be the set of immediate predecessors of job $i$ $(i=1,\ldots,n)$ in graph $G_P'$. Assume that the jobs are renumbered so that $i<j$ for each arc $(i,j)$ of $G_P'$. The Lagrangean problem is solved by a dynamic programming recursion defined on $F_i(t)$ which represents the minimum total cost of scheduling job $i$ and its predecessors to be completed not later than time $t$, where the contributions to the total cost for each job are its weighted tardiness and a multiplier for each time interval in which it is scheduled. The lower bound is given by

$$\mathrm{LB}_{\mathrm{LR}}(\mu)=F_n(P)-\sum_{t=1}^{P}\mu_t$$

(recall that since $G'_P$ is connected, job $n$ has all other jobs as its predecessors), where $F_n(P)$ is found from the recursion equations

$$F_i(t) = \begin{cases} \infty, & \text{for } t = 0, \ldots, C_i^E - 1, \\ \min\left\{F_i(t-1), \sum\limits_{k \in \beta_i} F_k(t-p_i) \right. & \\ \quad \left. + w_i \max\{t - d_i, 0\} + \sum\limits_{t'=t-p_i+1}^{t} \mu_{t'}\right\}, & \text{for } t = C_i^E, \ldots, C_i^L, \\ F_i(C_i^L), & \text{for } t = C_i^L + 1, \ldots, P. \end{cases}$$

The subgradient optimization method is used to find the vector of multipliers $\mu$. For each iteration of this method, the Lagrangean problem is solved in $O(nP)$ time. The usual way to implement a bound which requires subgradient optimization in a branch and bound algorithm is to perform a fairly large number of subgradient optimization iterations at the root node of the search tree (100 in our computational experiments), whereas at other nodes the multipliers are updated from their values at the parent node by performing a smaller number of iterations (10 in our computational experiments).

### *4.5. Dynamic programming state-space relaxation*

In this section, it is shown how the dynamic programming state-space relaxation method is used to derive a lower bound. This method uses the techniques developed by Christofides et al. [3] for routing problems and by Abdul-Razaq and Potts [1] for a single machine scheduling problem. To derive this lower bound, consider the dynamic programming formulation of the problem given by recursion (1). Solving recursion equations (1) is equivalent to finding the shortest path in a *state-space graph* $G_S$. The nodes of $G_S$ correspond to states $(R, j)$ (where $j \in R$ and where $R$ and $R - \{j\}$ are feasible sets) and the arcs correspond to decisions whereby the transition to a new state from a previous state is achieved by the scheduling of a job. The length of the arc directed to node $(R, j)$ is $w_j \max\{\sum_{k \in R} p_k - d_j, 0\}$.

Unfortunately, as $n$ increases the numbers of nodes in $G_S$ increases rapidly and the space required for the storage of values $f(R, j)$ (or values $f(R, *)$) exceeds available storage. Instead of using recursion (1) directly to solve the problem, we derive from it a lower bounding scheme that is used in a branch and bound algorithm.

Our lower bound is derived from the dynamic programming state-space relaxation method in which the recursion is performed on a suitably relaxed state-space containing fewer states than in the original formulation. This is achieved by mapping the first state variables representing feasible sets of jobs onto new first state variables representing the total processing time of jobs in the set, i.e., a state $(R, j)$ is mapped onto a state $(\sum_{i \in R} p_i, j)$. The relaxed problem is solved by finding $\min_{j \in S}\{f'(P, j)\}$ using the recursion equations

$$f'(t,j)=\begin{cases}\infty, & \text{for } t=1,\ldots,C_j^E-1,\ C_j^L+1,\ldots,P,\\ \min\limits_{i\in S-\{j\}}\{f'(t-p_j,i) & \\ \quad +w_j\max\{t-d_j,0\}\}, & \text{for } t=C_j^E,\ldots,C_j^L,\end{cases}\tag{15}$$

that are initialized by setting $f'(t,j)=\infty$ for $t<0$ and $f'(0,j)=0$ $(j=1,\ldots,n)$.

Let $G_S'$ be the relaxed state-space graph corresponding to recursion (15). To verify that $\min_{j\in S}\{f'(P,j)\}$ is a valid lower bound, we observe that for any path in $G_S$, there is a corresponding path in $G_S'$ having the same length. Thus, the shortest path in $G_S'$ cannot be longer than the shortest path by $G_S$. A formal proof is given by Abdul-Razaq and Potts.

In contrast to the state-space graph $G_S$, the relaxed state-space graph $G_S'$ contains paths which do not correspond to feasible sequences. Such paths correspond to "sequences" in which some jobs appear more than once while others do not appear. However, the same job cannot appear in adjacent positions. Because recursion (15) is likely to generate an infeasible sequence, the resulting lower bound is weak. We describe next how this weakness may be overcome by a method that attempts to force the shortest path in the relaxed state-space graph to define a feasible sequence.

We define a *penalty* $\lambda_i$ as an additional cost that is incurred when job $i$ $(i=1,\ldots,n)$ is scheduled. Thus, the total cost of scheduling job $i$ to be completed at time $t$ is $w_i\max\{t-d_i,0\}+\lambda_i$. The introduction of penalties yields an equivalent problem since the cost of every schedule is increased by $\sum_{i=1}^n\lambda_i$, i.e., the length of every path in the state-space graph $G_S$ is increased by $\sum_{i=1}^n\lambda_i$. However, for the relaxed state-space graph $G_S'$ different paths are increased in length by different amounts when penalties are introduced and, consequently, the shortest path may change. Ideally, penalties would be chosen to force the shortest path in the relaxed state-space graph to define a feasible sequence which would then be optimal. These penalties are analogous to the multipliers used in Lagrangean relaxation.

We now give precise details of how penalties are used. If $\lambda=(\lambda_1,\ldots,\lambda_n)$ is a given vector of penalties, then the original problem is solved by computing $\min_{j\in S}\{f(S,j;\lambda)\}-\sum_{i=1}^n\lambda_i$ from the recursion equations

$$f(R,j;\lambda)=\min_{i\in R-\{j\}}\left\{f(R-\{j\},i;\lambda)+w_j\max\left\{\sum_{k\in R}p_k-d_j,0\right\}+\lambda_j\right\}$$

that are initialized by setting $f(\emptyset,j;\lambda)=0$ for $j=1,\ldots,n$. By mapping a state $(R,j)$ onto a state $(\sum_{i\in R}p_i,j)$ a relaxed problem is obtained which is solved by finding $\mathrm{LB}_{\mathrm{SSR}}(\lambda)=\min_{j\in S}\{f'(P,j;\lambda)\}-\sum_{i=1}^n\lambda_i$ from the recursion equations

$$f'(t,j;\lambda)=\begin{cases}\infty, & \text{for } t=1,\ldots,C_j^E-1,\ C_j^L+1,\ldots,P,\\ \min\limits_{i\in S-\{j\}}\{f'(t-p_j,i;\lambda)+w_j\max\{t-d_j,0\}+\lambda_j\}, & \\ \qquad \text{for } t=C_j^E,\ldots,C_j^L,\end{cases}\tag{16}$$

that are initialized by setting $f'(t, j; \lambda) = \infty$ for $t < 0$ and $f'(0, j; \lambda) = 0$ $(j = 1, \ldots, n)$.

To solve recursion equations (16) and to execute a backtracking procedure that finds the corresponding schedule, it is sufficient to store quantities $f'(t, *; \lambda)$, $f'(t, **; \lambda)$, $e(t, *; \lambda)$ and $e(t, **; \lambda)$ for $t = 0, \ldots, P$, where $f'(t, *; \lambda)$ and $f'(t, **; \lambda)$ represent the smallest and second smallest values chosen from $\{f'(t, 1; \lambda), \ldots, f'(t, n; \lambda)\}$ and $e(t, *; \lambda)$ and $e(t, **; \lambda)$ are the job indices that give these smallest and second smallest values. (Full details about the implementation are given in [1].) Thus, recursion (16) is solved using $4(P+1)$ words of storage.

The subgradient optimization method is used to find the vector of penalties $\lambda$. For each iteration of this method, the dynamic programming recursion equations are solved in $O(nP)$ time. The lower bound $LB_{SSR}$ is implemented in a branch and bound algorithm in a similar way to $LB_{LR}$: at the root node, a large number (100 in our computational experiments) of subgradient optimization iterations are performed, whereas the penalties are updated from their values at the parent node by performing a fairly small number (10 in our computational experiments) of iterations at other nodes of the search tree.

## 5. Branch and bound algorithms

This section describes the general framework of a branch and bound algorithm which may employ any of the lower bounding schemes described in the previous section. The branching rule, search strategy and dominance rules are identical to those used by Potts and Van Wassenhove [15].

Initially, the precedence graph $G_P$ is constructed from Emmons' dominance rules as described in Section 2. Also at the root node of the search tree two *heuristic methods* are used to schedule the jobs. The better of the two heuristic sequences is used to provide an initial upper bound. The first heuristic method selects a job with no successors in $G_P$ to be sequenced in the last unfilled position in the sequence: when there is a choice, a job is chosen for which its weighted tardiness when sequenced in this last position is as small as possible. The selected job is deleted and the process is repeated until all jobs are scheduled. The second heuristic is a straightforward generalization to the case of weighted tardiness of the method of Wilkerson and Irwin [25].

The branch and bound algorithms use a *backward sequencing branching rule* which generates a search tree for which nodes at level $l$ correspond to final partial sequences in which jobs are sequenced in the last $l$ positions. A *newest active node search* selects a node from which to branch.

A branch of the search tree in which a job is added to a final partial sequence is discarded unless all successors of that job in $G_P$ appear in the final partial sequence. Further nodes are eliminated using Elmaghraby's lemma: if in any subproblem it is possible to sequence a job last so that it has zero tardiness, then a single node is added to the search tree which sequences that job last in the subproblem.

A further attempt is made to eliminate nodes using two tests which are based on the *dominance theorem of dynamic programming*. The first of these tests uses an *adjacent job interchange* to compare the sum of weighted tardiness for the two jobs most recently added to the final partial sequence with the corresponding sum when these two jobs are interchanged in position: if the former sum is larger than the latter, then the current node is eliminated, while if both sums are the same, some convention is used to decide whether the current node should be discarded. The second test uses the *job labelling* procedure of Schrage and Baker to construct an address for each subset of jobs that can form a final partial sequence of jobs which is consistent with the precedence graph $G_P$. Using the labelling scheme, we can easily check whether such a final partial sequence can be compared with one that has been previously generated and, if so, whether the current node is dominated. When it is not dominated, the total weighted tardiness of jobs of the current partial sequence is stored, replacing any previously stored quantity in that address. This second test is limited to those partial sequences with an address which does not exceed the number of words of available computer storage.

For all nodes that remain after the dominance tests are applied, we compute one of the lower bounds described in Section 4. If the lower bound for any node is greater than or equal to the smallest of the previously generated upper bounds, then that node is discarded.

## 6. Computational experience

This section compares the performance of the dynamic programming algorithms of Schrage and Baker, and Lawler, denoted by DPSB and DPLAW respectively, and the branch and bound algorithms BBLIN, BBEXP, BBLR and BBSSR which use the linear, exponential, Lagrangean relaxation and dynamic programming state-space relaxation lower bounds respectively. The transportation bound was not tested in a branch and bound algorithm because of the poor results obtained by Van Wassenhove and Gelders [24] for the problem in which the objective is to minimize the sum of total weighted tardiness and total weighted completion time. It appears that its computational requirements are too high relative to the quality of the bound which is generated. Furthermore, the assignment bound and its generalization proposed by Picard and Queyranne were not tested in a branch and bound algorithm either, since results of Schrage and Baker show that algorithm DPSB is far superior.

The algorithms were tested on problems with 20, 30, 40 and 50 jobs that were generated as follows. For each job $i$, an integer processing time $p_i$ and an integer weight $w_i$ were generated from the uniform distribution [1,10]. Problem "hardness" is likely to depend on parameters RDD and TF called the *relative range of due dates* and the *average tardiness factor*. Having computed $P = \sum_{i=1}^{n} p_i$ and selected a value of RDD and TF from the set $\{0.2,0.4,0.6,0.8,1.0\}$, an integer due date $d_i$ from the uniform distribution $[P(1 - \mathrm{TF} - \mathrm{RDD}/2), P(1 - \mathrm{TF} + \mathrm{RDD}/2)]$

was generated for each job $i$. Three problems were generated for each of the 25 pairs of values of RDD and TF, yielding 75 problems for each value of $n$. Note that these problems are generated in the same way as those of Potts and Van Wassenhove [15] except that in the latter case integer processing times are generated from the uniform distribution [1,100]. Our method yields smaller processing times with the result that the branch and bound algorithms that use the lower bounds $LB_{LR}$ and $LB_{SSR}$ which require pseudopolynomial time are favoured by these test problems. Also, these problems with a small number of distinct processing times tend to be easier because the conditions of Emmons' dominance rules are easier to fulfil.

The algorithms were coded in FORTRAN V and run on a CDC 7600 computer. For the four branch and bound algorithms, whenever a problem was not solved within a time limit of 60 seconds, computation was abandoned for that problem. Also, due to the storage limits of the two dynamic programming algorithms, problems are unsolved if the sum of labels for algorithm DPSB exceeds 48 000 and if the number of feasible sets with cardinalities differing by at most one for algorithm DPLAW exceeds 16 000. Algorithms BBLR and BBSSR were not tested on the problems with 40 and 50 jobs because of the discouraging results obtained for $n = 30$.

Results comparing the performances of the algorithms are given in Tables 1 and 2. For each value of $n$, Table 1 lists the median computation time in seconds required by each algorithm to solve the test problems and also, when any exist, gives the number of unsolved problems. For algorithm DPSB when $n = 40$ and $n = 50$ and for algorithm DPLAW when $n = 50$, however, because over half of the problems are unsolved the median cannot be computed. Table 2 lists the median sum of labels for algorithm DPSB, the median numbers of feasible sets generated for algorithm DPLAW (excluding the entry for $n = 50$ where the median cannot be computed) and the median numbers of nodes in the search tree for the branch and bound algorithms.

Before giving an overall comparison, it is appropriate to discuss the algorithms in pairs. The dynamic programming algorithms are compared first, followed by the branch and bound algorithms BBLIN and BBEXP which use quickly computed lower bounds. Lastly, algorithms BBLR and BBSSR that use tighter lower bounds which require pseudopolynomial time are discussed.

We first observe from Table 1 that although algorithm DPLAW is able to solve several problems that are unsolved when algorithm DPSB is applied, computation

Table 1. Median computation time in seconds and numbers of unsolved problems

| $n$ | DPSB | DPLAW | BBLIN | BBEXP | BBLR | BBSSR |
|---|---|---|---|---|---|---|
| 20 | 0.03 | 0.06 | 0.03 | 0.05 | 2.48 | 1.65 |
| 30 | 0.15: 12 | 0.47: 2 | 0.14 | 0.24 | 6.91: 2 | 7.96: 3 |
| 40 | - : 43 | 4.75: 25 | 0.67: 4 | 1.28: 3 | - | - |
| 50 | - : 45 | - : 39 | 2.03: 16 | 5.05: 19 | - | - |

Table 2. Median sums of labels, numbers of feasible sets and numbers of nodes

| $n$ | DPSB | DPLAW | BBLIN | BBEXP | BBLR | BBSSR |
|---|---|---|---|---|---|---|
| 20 | 504 | 434 | 91 | 104 | 53 | 28 |
| 30 | 7820 | 3799 | 319 | 456 | 135 | 112 |
| 40 | 73253 | 33873 | 1162 | 1781 | - | - |
| 50 | 422340 | - | 2717 | 4128 | - | - |

times are generally larger than those for algorithm DPSB. These results are in accordance with those obtained by Potts and Van Wassenhove [16] for the total tardiness problem but at variance with those given by Kao and Queyranne for the assembly line balancing problem where Lawler's algorithm is found to require less computation time than the Schrage–Baker algorithm. It would be misleading to suggest from our results that one of these algorithms is clearly superior to the other since algorithm DPSB is fast and easy to code whereas algorithm DPLAW is able to solve larger problems using the same amount of core storage.

Tables 1 and 2 indicate that algorithm BBLIN is superior to algorithm BBEXP. However, the results for $n=40$ show that there is one less unsolved problem for BBEXP, so our new exponential bound does have some merits. At the root node of the search tree where the golden section search is used to find $\alpha^*$, the bound $LB_{EXP}$ is sometimes substantially better than $LB_{LIN}$ for problems with small TF ($TF \leq 0.6$) and tends to be only slightly worse for problems with large TF ($TF \geq 0.8$). Unfortunately, the bounds are not tight enough to justify the use of the golden section search at each node of the search tree. In spite of much initial experimentation with various heuristic methods to compute values of $\alpha$ within the search tree, we are unable to find a method which yields smaller search trees than those generated by algorithm BBLIN.

We next observe from Table 1 that algorithm BBLR is slightly better than algorithm BBSSR for the problems tested, even though the median numbers of nodes in the search tree are larger. However, in both cases computation times are large. Results of Abdul-Razaq and Potts for the problem in which jobs have costs for earliness as well as for tardiness, where there are no dominance rules analogous to those of Emmons, show that algorithm BBSSR is superior. The most likely explanation of the difference is that the lower bound $LB_{LR}$ uses some of the constraints of $G_P$ whereas $LB_{SSR}$ relaxes all such constraints after the earliest and latest completion times are computed. Initial experiments with a modified version of $LB_{SSR}$ which retains some of the precedence constraints of $G_P$, indicate that the improvement to the lower bound is insufficient to compensate for its extra computational requirements.

Lastly, we give an overall comparison of the algorithms. The branch and bound algorithms BBLIN and BBEXP are the most efficient and are able to solve effectively problems with up to 40 jobs. Computational results indicate that algorithm BBLIN is superior. The dynamic programming algorithms require too much core storage

to compare favourably with these branch and bound algorithms although computation times are small for algorithm DPSB. The tighter lower bounds employed in algorithms BBLR and BBSSR successfully limit the size of search trees but by an insufficient amount to justify their heavy computational requirements. For all algorithms, unsolved problems tend to lie in those classes which have traditionally been considered the hardest.

## 7. Concluding remarks

In this paper we discuss and compare algorithms for the single machine total weighted tardiness problem. The branch and bound algorithm of Potts and Van Wassenhove [15] (BBLIN) which obtains a lower bound from a linear function of completion times problem is the most efficient and is able to solve problems with up to 40 jobs. A similar algorithm (BBEXP) which replaces the linear function with an exponential function also yields reasonable results. For the other branch and bound algorithms which use Lagrangean relaxation of machine capacity constraints (BBLR) and dynamic programming state-space relaxation (BBSSR), the computational requirements of the lower bounds are too time consuming to yield a competitive algorithm. Results for the Schrage-Baker algorithm (DPSB) show that dynamic programming algorithms can yield small computation times, although for larger problems dynamic programming is limited by computer core storage requirements, even when special attempts are made to minimize storage (DPLAW).

A theoretical result of Rothkopf and Smith [19] shows that no further scheme that uses the ideas of the linear and exponential function of completion time bounds can be derived. To solve larger problems, a tighter lower bound than that obtained from a linear function of completion times is needed. Ideally, it would require polynomial time, although a noniterative pseudopolynomial scheme would not be ruled out. Unfortunately, there is no obvious way to approach the derivation of lower bounds having these desired characteristics.

## Acknowledgment

The research by the first author was sponsored by a grant from Basra University, Iraq.

## References

[1] T.S. Abdul-Razaq and C.N. Potts, Dynamic programming state-space relaxation for single machine scheduling, J. Oper. Res. Soc. 39 (1988) 141–152.

[2] K.R. Baker and J.B. Martin, An experimental comparison of solution algorithms for the single-machine tardiness problem, Naval Res. Logist. Quart. 21 (1974) 187–199.

[3] N. Christofides, A. Mingozzi and P. Toth, State-space relaxation procedures for the computation of bounds to routing problems, Networks 11 (1981) 145–164.
[4] S.E. Elmaghraby, The one-machine sequencing problem with delay costs, J. Indust. Eng. 19 (1968) 105–108.
[5] H. Emmons, One-machine sequencing to minimize certain functions of job tardiness, Oper. Res. 17 (1969) 701–715.
[6] M.L. Fisher, A dual algorithm for the one-machine scheduling problem, Math. Programming 11 (1976) 229–251.
[7] M.L. Fisher, The Lagrangian relaxation method for solving integer programming problems, Management Sci. 27 (1981) 1–18.
[8] L. Gelders and P.R. Kleindorfer, Coordinating aggregate and detailed scheduling in the one-machine job shop I: Theory, Oper. Res. 22 (1974) 46–60.
[9] L. Gelders and P.R. Kleindorfer, Coordinating aggregate and detailed scheduling in the one-machine job shop II: Computation and structure, Oper. Res. 23 (1975) 312–324.
[10] A.M Geoffrion, Lagrangean relaxation for integer programming, Math. Programming Stud. 2 (1974) 82–114.
[11] E.P.C. Kao and M. Queyranne, On dynamic programming methods for assembly line balancing problems, Oper. Res. 30 (1982) 375–390.
[12] E.L. Lawler, On scheduling with deferral costs, Management Sci. 11 (1964) 280–288.
[13] E.L. Lawler, Efficient implementation of dynamic programming algorithms for sequencing problems, Rept. BW 106, Mathematisch Centrum, Amsterdam (1979).
[14] J.-C. Picard and M. Queyranne, The time dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling, Oper. Res. 26 (1978) 86–110.
[15] C.N. Potts and L.N. Van Wassenhove, A branch and bound algorithm for the total weighted tardiness problem, Oper. Res. 33 (1985) 363–377.
[16] C.N. Potts and L.N. Van Wassenhove, Dynamic programming and decomposition approaches for the single machine total tardiness problem, European J. Oper. Res. 32 (1987) 405–414.
[17] A.H.G. Rinnooy Kan, B.J. Lageweg and J.K. Lenstra, Minimizing total costs in one-machine scheduling, Oper. Res. 23 (1975) 908–927.
[18] M.H. Rothkopf, Scheduling independent tasks on parallel processors, Management Sci. 12 (1966) 437–447.
[19] M.H. Rothkopf and S.A. Smith, There are no undiscovered priority index rules for minimizing total delay costs, Oper. Res. 32 (1984) 451–456.
[20] L. Schrage and K.R. Baker, Dynamic programming solution of sequencing problems with precedence constraints, Oper. Res. 26 (1978) 444–449.
[21] J. Shwimer, On the *N*-job, one-machine, sequence-independent scheduling problem with tardiness penalties: A branch-and-bound solution, Management Sci. 18 (1972) B301–313.
[22] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logist. Quart. 3 (1956) 59–66.
[23] V. Srinivasan, A hybrid algorithm for the one-machine sequencing problem to minimize total tardiness, Naval Res. Logist. Quart. 18 (1971) 317–327.
[24] L. Van Wassenhove and L. Gelders, Four solution techniques for a general one machine scheduling problem: A comparative study, European J. Oper. Res. 2 (1978) 281–290.
[25] L.J. Wilkerson and J.D. Irwin, An improved algorithm for scheduling independent tasks, AIIE Trans. 3 (1971) 239–245.